processing_functions

These are some functions that allow for processed analysis of results, visualizations, and manipulation of data frames. Packages are loaded so that these functions can be used after sourcing this file. See README on mathematical model page for examples.

There is quite a bit of legacy code in here allowing for models without the latent phase, which is not consistent with the biology of CMV. Same with AUCData (see CMV_Models.pdf for note).

Packages

```
library(dplyr)
library(deSolve)
library(ggplot2)
library(gridExtra)
library(scales)
library(RColorBrewer)
library(doParallel)
library(reshape2)
library(knitr)
```

Functions

1. make_plot_data

takes fitted parameters and raw data from a single subject's results to generate long form of simulated time series (for ggplot2 style plotting). Will cycle through a set of models in fitOutput if multiple results are included. Can be called independently but also used by make_subject_plots.

```
#because of start_day, the model starts before data, first_time < 0 would plot that
make_plot_data = function(fitOutput, fitData = NULL, parallel = F, first_time = F){

  AUCData = NULL

  if(is.null(fitData)) {
    max_time = 800
    } else {
      fitData = fitData %>% dplyr::mutate(days_model = days2)
      max_time = max(fitData$days_model)
      fitData$count2 = fitData$count
      fitData$count2[which(fitData$count2 == 0)] <- NA
    }

  simData = plyr::ldply(unique(fitOutput$model), function(modelIn){

    submodelparms = subset(fitOutput, model == modelIn)
    if(first_time) return_time = -round(submodelparms$start_day, 1) else return_time = 0
    times = seq(-round(submodelparms$start_day, 1), max_time, 0.1)

    if(grepl("latent", modelIn)){
          if(grepl("linear", modelIn)){
            init = with(submodelparms, c(S = K, I0 = 1, I = 0, V = 0))
            }else init = with(submodelparms, c(S = K, I0 = 1, I = initI, V = initV, Tcell = 0))
    }else{ #no latent stage
      if(modelIn == "CMVModel_linear"){
        init = with(submodelparms, c(S = K, I = initI, V = initV))
        } else init = with(submodelparms, c(S = K, I = initI, V = initV, Tcell = 0))
    }

    out = as.data.frame(lsoda(init, times, get(modelIn),
                              unlist(select(submodelparms %>% ungroup(), -model, -PatientID2)), AUCData
    out = subset(out, time >= return_time)
    data.frame(
      model = modelIn,
      days_model = out$time,
      count = log10(out$V),
      S = out$S,
      I0 = if(!is.na(init["I0"])) out$I0 else NA,
      I = out$I,
      V = out$V,
      Tcell = if(!is.na(init["Tcell"])) out$Tcell else NA
    )

  }, .parallel = parallel)
  if(is.null(fitData)) simData
  else{
    fitData$model = "data"
    fitData$S = NA
    fitData$I0 = NA
    fitData$I = NA
    fitData$V = NA
    fitData$Tcell = NA

    combinedData = rbind(simData, fitData[,c("model", "days_model", "count" , "S", "I0", "I", "V", "Tcel
    combinedData$PatientID2 = fitData$PatientID2[1]

    combinedData
  }

}
```

2. plot_model

plots the simulation data from make_plot_data as a line graph. A line for each model type and raw data is shown as points. Can be called with data from make_plot_data; also used by make_subject_plots. Print = T means the function call will plot the graph, otherwise the function returns the ggplot object.

```
plot_model = function(simulation_data, print = T){
  simulation_data$model2 = sapply(1:dim(simulation_data)[1], function(i)
    tail(strsplit(as.character(simulation_data$model[i]), "CMVModel_")[[1]], 1))
  pl = ggplot() +
    geom_point(data = subset(simulation_data, model == "data"), aes(x = days_model, y = count), colour =
    geom_line(data = subset(simulation_data, model != "data"),
              aes(x = days_model, y = count, colour = model2, linetype = model2), size = 1.5, alpha = 0
    scale_x_continuous("Time (days)", limits = c(min(simulation_data$days_model), 775), breaks = 100*0:
    scale_y_continuous("Log10 CMV DNA Conc.", limits = c(0, 9), breaks = 0:9) +
    scale_colour_discrete("") +
    scale_linetype_discrete("") +
    theme(legend.position = "top",
          text = element_text(family = "Times"),
          title = element_text(size = 6),
          legend.text = element_text(size = 8),
          legend.key.size = unit(0.5, "cm")) +
    ggtitle(simulation_data$PatientID2[1])

  if(print) print(pl)
  else pl

}
```

3. make_subject_plots

Takes all fit output and CMV data and calls make_plot_data subject by subject to create simulation data. If return_data = T then returns this giant data.frame. Else it then calls plot_model for each simulation set by subject id creating a list of plots. Save_out = T then it will use marrangeGrob in ggsave to create pdf of the plots (one plot per page). Otherwise it will return the list of the plots. TODO - also make "clearance plots" which shows the effect of the immune system (the rate of decay of I or V for each model).

```
#because of start_day, the model starts before data, first_time = T would plot that
make_subject_plots = function(fitOutput, allCMVData, parallel = F,
                              simulation_data = NULL, save_out = T, dir_set = NULL, return_data = F, fi

  if(is.null(simulation_data)){
    simulation_data = plyr::ldply(unique(fitOutput$PatientID2), function(id){

      subFitOutput = subset(fitOutput, PatientID2 == id)
      subRawData = subset(allCMVData, PatientID2 == id)

      make_plot_data(subFitOutput, subRawData, parallel = parallel, first_time = first_time)
    })
  }

  if(return_data) return(simulation_data)

  plots = plyr::llply(unique(simulation_data$PatientID2), function(id){

    subPlotData = subset(simulation_data, PatientID2 == id)

    plot_model(subPlotData, print = F)

    })
  if(is.null(dir_set)) dir = paste(Sys.Date())
    else dir = dir_set

  if(save_out) {
    ggsave(paste(dir, ".pdf", sep = ""), do.call(marrangeGrob, c(plots, list(nrow = 1, ncol = 1))))
    #ggsave(paste(dir, "_clearance.pdf", sep = ""), do.call(marrangeGrob, c(plots_clr, list(nrow = 1, n
  }
    else plots

}
```

4. quickplot

takes one subjects results for one model and plots them. Will overlay rawData points if specified.

```
quickplot = function(fitOutput, rawData = NULL){
  #one model only
  modelIn = fitOutput$model
  submodelparms = fitOutput
  if(length(modelIn) > 1) return("only input one model")

  if(grepl("latent", modelIn)){
        if(grepl("linear", modelIn)){
          init = with(submodelparms, c(S = K, I0 = 1, I = 0, V = 0))
          }else init = with(submodelparms, c(S = K, I0 = 1, I = initI, V = initV, Tcell = 0))
  }else{ #no latent stage
    if(modelIn == "CMVModel_linear"){
      init = with(submodelparms, c(S = K, I = initI, V = initV))
      } else init = with(submodelparms, c(S = K, I = initI, V = initV, Tcell = 0))
  }

  times = seq(-round(fitOutput$start_day, 2), 800, 0.1)

  out = as.data.frame(lsoda(init, times, get(modelIn), unlist(select(fitOutput, -model, -PatientID2))))

  plot = ggplot(data = out, aes(x = time, y = log10(V))) +
    geom_line() +
    ggtitle(paste(fitOutput$PatientID2, fitOutput$model))

  if(is.null(rawData)) print(plot)
  else {
    rawData = rawData %>% dplyr::mutate(days_model = days2)
    print(plot + geom_point(data = rawData, aes(x = days_model, y = count)))
  }
}
```

5. parameter_plot

Takes a list of parameters and output and plots them as a boxplot, default prints mean square error

```
parameter_plot = function(output_data, parameter_list = "mse"){

  parameter_fits = reshape2:::melt.data.frame(output_data, measure.vars = parameter_list,
                                              value.name = "value", variable.name = "parameter")

  ggplot(data =  subset(parameter_fits),
         aes(x = parameter, y = log10(value), colour = model)) +
    geom_boxplot() +
    scale_y_continuous(breaks = seq(-10, 8 , 2)) +
    scale_x_discrete("Parameters")
}
```

6. data_parameter_plot

This function plots the data with model fit in one plot and the parameter values next to it

```
#this must contain be passed the already made simulation data
data_parameter_plot = function(fitOutput, simulation_data, parameter_list){

  plots = plyr::llply(unique(simulation_data$PatientID2), function(id){

    subPlotData = subset(simulation_data, PatientID2 == id)
    subOutput = subset(fitOutput, PatientID2 == id)

    #makes the data and model sim plots
    plot_data = plot_model(subPlotData, print = F)


    #now make the parameter plots
    parameter_fits = reshape2:::melt.data.frame(subOutput, measure.vars = parameter_list,
                                              value.name = "value", variable.name = "parameter")

    fakeinput = rep(NA, length(parameter_fits$parameter))
    names(fakeinput) = as.character(parameter_fits$parameter)

    parameter_fits$lower = sapply(1:length(fakeinput), function(i) if(names(fakeinput)[i] == "mse") retu
                                  else return(boundary_set(fakeinput[i], "lower")))
    parameter_fits$upper = sapply(1:length(fakeinput), function(i) if(names(fakeinput)[i] == "mse") retu
                                  else return(boundary_set(fakeinput[i], "upper")))

    R0data = subOutput %>% group_by(model) %>% summarize(R0label = paste("R0 = ", round(R0, 2)))

    plot_parms = ggplot(data =  subset(parameter_fits),
         aes(x = parameter, y = log10(value), colour = model)) +
      geom_errorbar(aes(ymin = lower, ymax = lower), colour = "black", width = 0.5, linetype = "dashed"
      geom_errorbar(aes(ymin = upper, ymax = upper), colour = "black", width = 0.5, linetype = "dashed"
      geom_point(size = 3, alpha = 0.5) +
      scale_y_continuous(breaks = seq(-10, 10, 2)) +
      scale_x_discrete("Parameters") +
      geom_text(data = R0data, aes(label = R0label, x = -Inf, y = c(10)),
                vjust = 1.2, hjust = -.2) +
      theme(legend.position = "top",
        text = element_text(family = "Times"),
          title = element_text(size = 6),
          legend.text = element_text(size = 8),
          legend.key.size = unit(0.5, "cm"))

    return(plot_grid(plot_data, plot_parms, nrow = 1, ncol = 2))
    })
  return(plots)

}
```

7. make_highres_sim

For more exact estimation of finite differences

```
make_highres_sim = function(fitOutput, parallel = F, first_time = F){

  plyr::ldply(unique(fitOutput$PatientID2), function(id){
      subFitOutput = subset(fitOutput, PatientID2 == id)

      plyr::ldply(unique(subFitOutput$model), function(modelIn){
            submodelparms = subset(subFitOutput, model == modelIn)

            if(first_time) return_time = -round(submodelparms$start_day, 1) else return_time = 0
            times = seq(-round(submodelparms$start_day, 1), 350, 0.1)

            if(grepl("latent", modelIn)){
                  if(grepl("linear", modelIn)){
                    init = with(submodelparms, c(S = K, I0 = 1, I = 0, V = 0))
                    }else init = with(submodelparms, c(S = K, I0 = 1, I = initI, V = initV, Tcell = 0))
            }else{ #no latent stage
              if(modelIn == "CMVModel_linear"){
                init = with(submodelparms, c(S = K, I = initI, V = initV))
                } else init = with(submodelparms, c(S = K, I = initI, V = initV, Tcell = 0))
            }

            out = as.data.frame(lsoda(init, times, get(modelIn), unlist(select(submodelparms %>% ungrou

            out = subset(out, time >= return_time)

            data.frame(
              PatientID2 = id,
              model = modelIn,
              days_model = out$time,
              count = log10(out$V),
              S = out$S,
              I0 = if(!is.na(init["I0"])) out$I0 else NA,
              I = out$I,
              V = out$V,
              Tcell = if(!is.na(init["Tcell"])) out$Tcell else NA
            )

          }, .parallel = parallel)
  })}
```